1. (AMENDED) A computerized wagering game apparatus, comprising:

a computerized game controller having a processor, memory, and nonvolatile storage, [and] the computerized game controller being operable to control [the] a computerized wagering game;

an operating system comprising: a system handler application operable to dynamically link with at least one gaming program object;

an Application Program Interface callable from the gaming program object and

an operating system kernel that executes the system handler application.

2. The computerized wagering game apparatus of claim 1, wherein the system handler application comprises an event handler.

3. The computerized wagering game apparatus of claim 1, wherein the system handler application comprises software having the ability when executed to:

unload a previous gaming program object if a previous object has been loaded;

load a new gaming program object; and

execute the new gaming program object.

4. (AMENDED) The computerized wagering game apparatus of claim 1, wherein data variables modified by the gaming program objects are stored in nonvolatile storage, and functions verify that the operating system or code for a shared object has not changed.

5. The computerized wagering game apparatus of claim 4, further comprising a game state device providing a variable name index to associated variable data locations within the nonvolatile storage.

6. The computerized wagering game apparatus of claim 4, wherein changing a data variable in nonvolatile storage causes execution of a corresponding callback function in the system handler application gaming program object.

7. The computerized wagering game apparatus of claim 1, wherein the computerized game controller comprises an IBM PC-compatible computer.

8. The computerized wagering game apparatus of claim 1, wherein the operating system kernel is a Linux operating system kernel.

9. The computerized wagering game apparatus of claim 8, wherein the Linux operating system kernel has at least one selected device handler disabled.

10. The computerized wagering game apparatus of claim 9, wherein the at least one selected device handler that is disabled is selected from the group consisting of a keyboard handler, an I/O port handler, a network interface handler, a storage device controller handler, and a I/O device handler.

11. The computerized wagering game apparatus of claim 1, wherein the system handler application comprises an API with functions callable from the gaming program objects.

12. The computerized wagering game apparatus of claim 11, wherein the API comprises functions that are specific to a computerized gaming apparatus.

13. (AMENDED) A method of managing data in a computerized wagering game apparatus via a system handler application, comprising:

loading a first program object <u>and providing an Application Program Interface</u>,

executing the first program object,

storing data variables in nonvolatile storage, such that a second program object later loaded can access the data variables in nonvolatile storage,

unloading the first program object, and

loading a second program object.

14. The method of claim 13, further comprising executing a corresponding callback function upon alteration of variable data in nonvolatile storage.

15. The method of claim 13, further comprising handling events via the system handler application.

16. A secure computerized wagering game system controlled by a general-purpose computer, comprising an operating system kernel that is customized to disable selected device handlers.

17. A secure computerized wagering game system controlled by a general-purpose computer comprising nonvolatile storage that stores program variables, such that loss of power does not result in loss of the state of the computerized wagering game system.

18. The secure computerized wagering game system of claim 17, further comprising at least one gaming program object, such that a single gaming program object is loaded and executed

at any one time but gaming program objects are operable to share data via the program variables in nonvolatile storage.

19. A machine-readable medium with instructions thereon, the medium being within a wagering apparatus, the instructions when executed operable to cause a computer to:

cause a system handler application to load and execute gaming program objects;

cause a loaded gaming program object to call up a library of functions provided by the system handler application;

load a first program object from the library,

execute the first program object,

store data variables in nonvolatile storage, such that a second program object in the library later loaded can access the data variables in nonvolatile storage,

unload the first program object, and

load the second program object.

20. The machine-readable medium of claim 19, with further instructions operable when executed to cause a computer to execute a corresponding callback function upon alteration of variable data in the nonvolatile storage.

21. The machine-readable medium of claim 19, with further instructions operable when executed to cause a computer to manage events via the system handler application.

22. (AMENDED) A machine-readable medium with instructions thereon, the instructions when executed operable to cause a computer to manage at least one gaming program object via a system handler application, the gaming program object calling up an Application Program Interface such that a single gaming program object is loaded and executed at any

one time but gaming program objects are operable to share data via the program variables in a nonvolatile storage.

23. The machine-readable medium of claim 22, with further instructions operable when executed to cause a computer to provide functions through an API that comprises a part of the system handler application.

47. (AMENDED) A method of modifying an operating system kernel, comprising at least one modification to obtain functionality selected from the group consisting of:

1) accessing user level code from ROM;

2) executing user level code from ROM;

3) zeroing out unused RAM;

4) testing and/or hashing the kernel <u>to verify that the operating system kernel or a code for a shared object has not changed</u>;

5) disabling selected device handlers.

48. The computerized wagering game of claim 1 wherein the operating system kernel executing on the computerized game controller comprises an element of a universal operating system also comprising a system handler.

49. The computerized wagering game of claim 1 wherein the non-volatile storage is controlled by a general-purpose computer, the non-volatile storage stores game data, and the storage of game data in the non-volatile storage preserves the state of the computerized wagering game system upon loss of power.

50. The computerized wagering system of claim 1 operating within a networked on-line system.

51. The computerized wagering system of claim 1 wherein the system controls a progressive meter.

52. (AMENDED) The computerized wagering system of claim 1 wherein an Application Program Interface is also dynamically linked [by] <u>from</u> the systems handler.

53. (AMENDED) The machine readable medium of claim 19 wherein the instructions when executed are operable to dynamically link an Application Program Interface to a [system handler] gaming program object.

54. (AMENDED) The machine readable medium of claim 22 wherein the instructions when executed are operable to dynamically link an Application Program Interface to a [system handler] gaming program object.

PLEASE ADD THE FOLLOWING NEW CLAIMS:

55. The computerized wagering game apparatus of claim 1 wherein the operating system operates a function to verify that the operating system kernel or a code for a shared object has not changed.

56. The method of managing data in a computerized wagering game apparatus via a system handler application according to claim 1 wherein a function is performed to verify that an operating system kernel or a code for a shared object has not changed.

57. The machine-readable medium with instructions thereon wherein a function is performed to verify that an operating system kernel or a code for a shared object has not changed

REMARKS CONCERNING THE AMENDMENTS

Antecedent basis for the amendments and new claims may be found generally in the 0specification and, for example, on pages 7-14.

REMARKS CONCERNING THE OFFICE ACTION

In the Office Action mailed on August 2, 2002, the U.S. PTO:

a) Objected to the Drawings as failing to show every feature recited in the claims. Accompanying this Amendment and Response are proposed drawing corrections requested on Page 2, line 1 through page 3, line 3.

b) Claims 1-4, 6, 10-13, 18-21,, 23-28, 30-33, 35, 38-43, 48 and 49 were rejected under 35 USC 103(a) as unpatentable over Bunnell (U.S. Patent No. 6,075,939).

c) Claims 7, 8, 14, 21 and 52-54 have been rejected under 35 USC 103(a) as unpatentable over Bunnell in view of David Rusling, *The Linux Kernel* (hereinafter "Rusling")

d) Claims 5 and 15 have been rejected under 35 USC 103(a) as unpatentable over Bunnell in view of Pascal.

e) Claims 9, 10, 17, 38, 44 and 47 have been rejected under 35 USC 103(a) as unpatentable over Bunnell in view of Pascal in further view of Bock (U.S. Patent No. 5,155,856) and Davis (U.S. Patent No. 6,401,208).

f) Claims 45, 46, 50 and 51 have been rejected under 35 U.S.C. 103(a) as unpatentable over Bunnell in view of Wiltshire (U.S. Patent No. 6,409,602).

RESPONSE TO OBJECTIONS TO DRAWINGS

Accompanying this Amendment and Response is a new Figure 3 that constitutes a bona fide attempt to provide proposed changes to the figures that will incorporate references to each of the elements identified in the Office Action as being required in Figures. Formal drawings will be submitted upon approval by the PTO or upon allowance of the application.

RESPONSE TO REJECTION

a) Claims 1-4, 6, 10-13, 18-21,, 23-28, 30-33, 35, 38-43, 48 and 49 were rejected under 35 USC 103(a) as unpatentable over Bunnell (U.S. Patent No. 6,075,939).

The rejection in the Office Action is believed to be fairly summarized as follows:
Bunnell et al. is asserted to show:

1) A system handler, executed by the operating system kernel, operable to dynamically link with at least one program object;

2) An operating system comprising a system handler;

3) A system handler comprising a plurality of device handlers;

4) A system handler that loads and executes program devices;

5) The kernel is modified to access user level code from ROM;

6) The kernel is modified to execute from ROM;

7) Kernel modifications are modular;

8) The system handler comprises APIs with functions callable from program objects;

9) The system handler can manage an event queue; and

10) The system handler loads and unloads program objects.

The rejection then lists thirteen (13) elements (identified as b-n elements) in the claims that the Examiner agrees are not shown by Bunnell et al. The rejection then asserts that the elements not disclosed by Bunnell et al. are "typical game element device methods employed on a PC."

In separate rejection under 35 U.S.C. 1039a), the Examiner then cites four references (Rusling, *The Linux Kernel,* ,http://www.tldp.org/LDP/tlk/tlk.htm> (1999), Pascal et al. (U.S. Patent 5,971,851), Bock et al. (U.S. Patent No. 5,155,856), and Davis (U.S. Patent No. 6,401,208) to show individual elements that are recited in dependent claims as obvious over the teachings of Bunnell et al. and the Official Notice taken by the Examiner. These rejections are respectfully traversed.

The rejection fails in a number of regards. The first level of traversal in the rejection is the failure to appreciate the complexity of the use of a system handler in **A GAMING SYSTEM ENVIRONMENT**, the differences between general PC usage and usage in a gaming system, and the unique aspects of the system recited in the claims as compared to the systems described in the prior art cited against the claims. Additionally, the above amendments to the independent claims emphasize features thought to be implicit in certain claims but now literally and clearly recited in the claims and finding antecedent basis in the original specification as filed. For example, the limitations of:

a) to verify that the operating system kernel or a code for a shared object has not changed; (e.g., Page 11, lines 14-18)
b) the gaming program object calling up an Application Program Interface; (Page 6, lines 4-16) and
c) cause a system handler application load and execute gaming program objects; cause a loaded gaming program object to call up a library of functions; load a first program object from the library, (Page 6, lines 4-16 and Page 10, lines 8-26).

The differences between the gaming system environment and the fields of use described by Bunnell et al. are first evident in the fact that Bunnell et al. does not show a game controller, game programs, and game objects. There is nothing in Bunnell et al. that directly ties that reference into the field of practice recited in the claims, not only in the preamble, but within functional limitations in the elements of the claims themselves. This substantive initial difference becomes extremely important with regard to the differences in the function and components of the system and methods recited in the claims of the present application.

To begin with, the underlying operations of the claimed system and the system of Bunnell with regard to the terminology of "dynamic linkage" are quite distinct. Claim 1, for example, specifically recites:

> "...a **system handler application operable to dynamically link with at least one gaming program object**..."

This is quite distinct from what is described in Bunnell et al. For example, looking at Figure 4 of Bunnell et al. (and reviewing the entire specification of Bunnell), the operation of the technology does not show a system handler application that "dynamically links with at least one gaming program object..." In fact, the Figure and the disclosure do not show dynamic linking, as recited in the claim, that performs at the same hierarchal level as that recited in the claim. Looking at Figure 4 of Bunnell, the dynamic linking recited in the claim would be above (e.g.—at a higher level, such as at an application level, or between an application and operating system level) all elements shown in that Figure, not at the lower level of the ROM BIOS kernel shown in Figure 4 and described in the text of Bunnell et al. Therefore, in addition to failing to provide any direct nexus into the gaming art, the actual performance of Bunnell et al. is excluded by the present claims, and the present claims require the performance of steps and the presence of recited functions that are not shown by Bunnell.

Reviewing Bunnell in the most positive light, it can be seen that the level of dynamic linkage is in fact substantively different than that recited in the claims, between the system handler and a gaming program object. For example, looking at the complete disclosure of Bunnell, wherever "dynamic" or "linkage" or "linking" appears in the specific or claims, the following portions of the Bunnell specification should be noted.

At column 7, lines 62-65, Bunnell states under "Alternate components" that there should be a "dynamic system call facility" which does not expand the method of performance specifically taught by Bunnell and as illustrated by Figure 4. This is a general reference, with no probative teaching, and no description of a system handler dynamically linked to a gaming program object.

Column 10, lines 55-67 describes dynamically establishing relationships and then refers to objects that have already been linked. This is dynamic linking for system calls, dynamically linking system components together. That is different from the type of dynamic linking of the present invention. Figure 4 of Bunnell shows two kernel components linked together, which defines the level of dynamic linkage, in a Posix compliant API. Posix is a standard for operating systems, and Bunnell teaches an implementation of the standard system. Posix has nothing to do with the "dynamic linkage" recited in the claims. Bunnell describes a dynamic system call, not a dynamic linkage. At column 10, lines 60-66, Bunnell merely describes dynamically establishing relationships between global functions and symbols. This is not a description of a system handler dynamically linked to a gaming program object.

<u>**Effects of the Amendments on Establishing Non-Obviousness**</u>

This is extremely material with respect to the new limitations added to the claims. As noted, the system of Bunnell calls up an API directly and selects programs to be executed out of the API. Some of the claims as originally presented and now more of the claims (claims 1-47 and 49-54, only claim 48 not having this limitation) emphasize this fundamental difference. This is not a trivial change. The gaming system must operate in this manner to enhance security, which is a primary obligation and requirement of gaming equipment. There is no suggestion in Bunnell for this modification specifically needed in the gaming industry because Bunnell has no concept of the requirements of the gaming industry. Applicants have not noted this specific feature in the four other secondary references cited in later rejections under 35 USC 103(a). As this feature is not disclosed in the references and as this feature is a unique advantage in the gaming environment that is not central to the primary reference (Bunnell), the claims (1-47 and 49-54) reciting this limitation are not obvious from the art cited in the rejection.

The difference of the capability and actual performance at "higher levels" within the software is well understood in the art in general terms, and is spelled out clearly in the practice of the present invention in the dynamic linking of the system handler to the casino gaming objects. Software is written in different hierarchies of content. As one progresses from kernel level, to user API level, to object level etc., the program becomes more abstract moving up the hierarchy of the software. The kernel actually talks directly to hardware, then hardware can use game functions API to execute the gaming performance. This is what is meant when this response refers to a higher level of performance as compared to the software and operating systems of the references, where they do not operate at that level of abstraction.

Claim 48 was also amended to recite that, in addition to the novel operation of the software in the execution of data and objects, the system operates <u>to verify that the operating system kernel or a code for a shared object has not changed</u>. Bunnell does not show the combination of the recited execution of software and the verification process. New claims 55-57 also recite this additional step.

In addition, the performance of a dynamic linkage, as recited in these claims, allows for dynamic unlinking. This means that we can unload a game object and replace it with another game object as recited in other claims. This functionality, as recited in claims 5, 19 21, 22, 23, 25, and 26. For example, the recitations in claim 19 includes:

19. A machine-readable medium with instructions thereon, the medium being within a wagering apparatus, the instructions when executed operable to cause a computer to:

cause a system handler application load and execute gaming program objects;

cause a loaded gaming program object to call up a library of functions;

load a first program object from the library,

execute the first program object,

store data variables in nonvolatile storage, such that a second program object in the library later loaded can access the data variables in nonvolatile storage,

unload the first program object, and

load the second program object.

This functionality is not taught, enabled or suggested by Bunnell et al., alone or in combination with all of the secondary references.

Column 11, lines 20-65 of Bunnell describe "Dynamic System Call Management," which again is not a disclosure of a system handler that dynamically links to a gaming program object. The term "dynamic linkage" as used in the present invention and as described in the specification and understood in the art defines a linkage above the user program level, which is above the kernel level shown in the Figure (4 of Bunnell et al.) and as described in the specification of Bunnell. In addition, the user program links in the shared objects are gaming specific, which is not shown by Bunnell and is not motivated by the four additional references cited in the rejection. In the practice of the present invention, the dynamic linkage is also used with the API. This is quite distinct from Bunnell's POSIX system call for the API. This is sufficiently distinct as to have warranted the filing of new claims 52-54 to clearly claim that distinct feature.

The five times that dynamic system calls are referred to by Bunnell et al. on column 12 (lines, 4-67) do not in any way contradict the earlier usage or expand on that usage to indicate to one skilled in the art that a system handler can be dynamically linked to a gaming program object. Similarly, the reference on Column 13, lines 22-26 to a Dynamic System Call Table does not in any way improve upon the deficiencies noted with regard to the disclosure of Bunnell. That reference, and none of the secondary references, shows the dynamic linking of a system handler application to a program game object. Nothing equivalent to that has been shown to be taught by Bunnell et al. or any of the secondary references used in the rejection.

It is therefore clear, that even above the thirteen deficiencies that the Examiner has noted in Bunnell, there is another, and even more egregious deficiency that has not been asserted to be obvious in view of the secondary references.

Turning now to the thirteen deficiencies in the Bunnell et al. reference, the mere number of differences that the Office Action attempts to take Official Notice of is in itself an indication of the extent of difference between the teachings of Bunnell et al. and the claimed invention. Additionally, the statement that these features are "typical game device methods implemented on a PC" is challenged, as is the general taking of Official Notice on these limitations. The use of PC's in the gaming industry (as opposed to games) is relatively new. The use of a PC to effect these procedures and provide those functions was not obvious to applicants at the time of filing the Application. The extensive amount of work needed to enable the practice of these functions on a PC clearly indicates that there is neither obviousness nor ability to take Office Notice of those limitations, particularly within a gaming environment.

It is important to note here the environment of the present invention in the gaming industry. Applicants do not deny that it has been possible to use modern PC's and Operating Systems to write games, as that has been accomplished by the inventors. However, it was not obvious or instructed in the prior art or available in inherent components in available PC's and Operating Systems to meet Gaming Regulations. Additionally, gaming systems require significant and critical security in their communication systems, both internally and with outside sources of communication. Failure to provide such security would fail to meet both Gaming Commission requirements and the security needs of casino operators. No such communication security is generally required or used in arcade games. As described and enabled in the specification, extensive modifications, enhancements and safeguards were needed to effect gaming and system designs that could meet the stringent regulation standards, and adapt PC's to the casino environment. Accomplishing this result with a system handler dynamically linked to gaming program objects was not taught by any of the references cited in the rejection, either alone or in combination. The system handler as recited in the claims also provides the API to gaming program objects for security reasons. This is quite distinct from dynamic linkages that have been used in prior art systems of record in the art used in the rejection (if any) in which a program object provides an API to the Application (as done by Bunnell). The procedure of the invention is not taught by Bunnel.

The prior art of record also does not recognize the benefits of the dynamic linking recited in the practice of the invention. Because objects are loaded and then unloaded before the next object is loaded, resources in the memory are preserved, which is always an issue in the use of computerbased systems. This can enable the system to work more efficiently and more rapidly because of this dynamic linking and memory space utilization.

c) Claims 7, 8, 14, 21 and 52-54 have been rejected under 35 USC 103(a) as unpatentable over Bunnell in view of David Rusling, *The Linux Kernel* (hereinafter "Rusling")

Insofar as Rusling is cited to show that Linux is a useful and advantageous operating system, that teaching is accepted. However, Rusling provides no teaching whatsoever to overcome each and every one of the deficiencies noted above with respect to the teachings of Bunnell alone. The rejection of claims 7, 8, 14, 21 and 52-54 must therefore fail for at least the reasons that the lack of teaching of the recited and claimed elements of the claims from which these claims are dependent are not shown by Bunnell and those deficiencies are not overcome by Rusling.

d) Claims 5 and 15 have been rejected under 35 USC 103(a) as unpatentable over Bunnell in view of Pascal.

Insofar as Pascal is cited to show that alternative operating systems that have call back functions in a **game machine** (not gaming machine), that teaching is accepted. However, Pascal provides no teaching whatsoever to overcome each and every one of the deficiencies noted above with respect to the teachings of Bunnell alone. The rejection of claims 7, 8, 14, 21 and 52-54 must therefore fail for at least the reasons that the lack of teaching of the recited and claimed elements of the claims from which these claims are dependent are not shown by Bunnell and those deficiencies are not overcome by Pascal.

e) Claims 9, 10, 17, 38, 44 and 47 have been rejected under 35 USC 103(a) as unpatentable over Bunnell in view of Pascal in further view of Bock (U.S. Patent No. 5,155,856) and Davis (U.S. Patent No. 6,401,208).

As noted above with respect to Pascal, the fundamental deficiencies and lack of teaching of limitations in the independent claims found in Bunnell are not corrected by Pascal. The citation of Bock and Davis for their specific elements of disclosure for dependent claims do not correct the deficiencies that were not addressed by Bunnell in view of Pascal. These claims are therefore patentable because of their ultimate dependence from unobvious independent claims. Even assuming that the teachings of Bock and Davis are accurate for what is asserted, they do not overcome the earlier deficiencies and cannot establish obviousness of the dependent claims.

Claims 45, 46, 50 and 51 have been rejected under 35 U.S.C. 103(a) as unpatentable over Bunnell in view of Wiltshire (U.S. Patent No. 6,409,602).

Similar to the arguments that the secondary references (Pascal, Rusling, Davis, and Bock) do not either overcome the deficiencies of Bunnell, Whitling does not teach the fundamental limitations of the claims and therefore cannot improve or correct the earlier rejection.

Bock et al. has been cited as showing zeroing-out unused registers, asserting that is what is recited in claims 9, 17, 44, and 47. That is not equivalent to what is done in the present invention by zeroing out memory. Actually, zeroing out the memory does not disable device handlers. The two elements that the rejection attempts to correlate are unrelated separate security procedures. This is specifically recited in new claims 52-55. The teaching of Bock et al. asserted by the Office Action to describe a method of zeroing out unused registers to provide security during booting up is not equivalent to "...an operating system kernel that is customized to disable selected device handlers..." recited in the claims and the disabling of selected device handlers. Those are immaterial to the booting up security described by Bock et al. Even more importantly, the methods described in Bock et al. are implemented by hardware. The device handlers recited in the claims of the present invention which zero out memory are implemented in software. This means that in the practice of the invention recited in the claims, there is a CPU that fetches instructions from memory, and it is those instructions that causes the CPU to zero out memory. This zeroing out is not performed during boot-up, but during actual operation of the machine.

Similarly, Pascal is cited against claims 5, 15 and 22 as showing the limitations of those claims wherein it is recited that:

"...a game state device providing a variable name index to associated variable data locations within the nonvolatile storage ..."
"...handling events via the system handler application ..." and
'...such that a single gaming program object is loaded and executed at any one time but gaming program objects are operable to share data via the program variables in a nonvolatile storage ..."

The deficiencies noted above for Bunnell et al. and these limitations are not described, rendered obvious or enabled by the showing of data call backs taught by Pascal.

SUMMARY OF ARGUMENTS

The primary reference not only fails to establish utility of the teachings of that reference in the field of gaming, but also, in addition to the thirteen elements and limitations of the invention acknowledged by the Office Action to be missing from the Bunnell et al. reference, that reference fails to describe, suggest or enable a system handler to dynamically link to at least one gaming